Certification Authorities Software Team (CAST)

Position Paper CAST-12

Guidelines for Approving
Source Code to Object Code Traceability

Completed December 2002

NOTE: **This position paper has been coordinated among the software specialists of certification authorities from the United States, Europe, and Canada. However, <u>it does not constitute official policy or guidance from any of the authorities</u>. This document is provided for educational and informational purposes only and should be discussed with the appropriate certification authority when considering for actual projects.**

Guidelines for Approving
Source Code to Object Code Traceability

Abstract:

Approval of aviation software to the guidance of DO-178B/ED-12B requires an applicant to assess the correspondence between source code and object code in certain circumstances. In particular, for Level A software for which structural coverage is performed on the source code, source code to object code traceability must be addressed (see paragraph 6.4.4.2b of DO-178B/ED-12B). Then, if the compiler generates object code that is not directly traceable to the source code, the applicant must identify that untraceable, compiler-generated object code and verify it. This paper provides guidelines for certification authorities (and their designees, if applicable) to assess an applicant's source code to object code traceability and verification activities and results, when a review of Level A software is performed. As such, the paper could have an impact on the applicants.

Key words:
modified condition/decision coverage (MC/DC), source code to object code traceability, structural coverage, source code, object code, compiler, Level A, verification

1. <u>INTRODUCTION</u>. Approval of aviation software to the guidance of DO-178B/ED-12B requires an applicant to assess the correspondence between source code and object code in certain circumstances. In particular, for Level A software for which structural coverage is performed on the source code, source code to object code traceability must be addressed (see paragraph 6.4.4.2b of DO-178B/ED-12B). Then, if the compiler generates object code that is not directly traceable to the source code, the applicant must identify that untraceable, compiler-generated object code and verify it. This paper provides guidelines for certification authorities (and their designees, if applicable) to assess an applicant's source code to object code traceability and verification activities and results, when a review of Level A software is performed. As such, the paper could have an impact on the applicants.

2. <u>RELATED PUBLICATIONS</u>.

a. Advisory Circular 20-115B "Radio Technical Commission for Aeronautics, Inc. Document RTCA/DO-178B."

b. RTCA/DO-178B and EUROCAE/ED-12B, "Software Considerations in Airborne Systems and Equipment Certification."

c. RTCA/DO-248B and EUROCAE/ED-94B, "Final Report for Clarification of DO-178B/ED-12B 'Software Consideration in Airborne Systems and Equipment Certification.'"

d. Joint Airworthiness Authority (JAA) Temporary Guidance Leaflet Number 4 (TGL No. 4).

3. BACKGROUND. Test coverage analysis addressed in section 6.4.4 of DO-178B/ED-12B is embodied in a number of the objectives of the document. The two components of this analysis are Requirements-Based Test Coverage Analysis in section 6.4.4.1 and Structural Coverage Analysis in section 6.4.4.2. The objectives for structural coverage analysis (Table A-7, objectives 5-8) only apply for software levels A, B, and C. The purpose of this analysis is to determine which code structure was not exercised by the requirements-based tests. Section 6.4.4.2b of DO-178B/ED-12B states: *"The structural coverage analysis may be performed on the Source Code, unless the software level is A and the compiler generates object code that is not directly traceable to Source Code statements. Then, additional verification should be performed on the object code to establish the correctness of such generated code sequences. A compiler-generated array-bound check in the object code is an example of object code that is not directly traceable to the Source Code."* Section 6.4.4.2b indicates that this structural analysis may be performed on the source code. For level A software, DO-178B/ED-12B section 6.4.4.2b indicates that additional verification should be performed if the compiler generates object code that is not directly traceable to the source code. The amount and type of additional verification activity needed and how source code can be shown to be directly traceable to object code has been inconsistently applied and interpreted by certification authorities and applicants. The use of the term compiler in this paper is meant to include the effects of linkers and libraries as well. This paper will clarify what evidences are needed to demonstrate compliance with section 6.4.4.2b in DO-178B/ED-12B.

4. SCOPE. This paper specifically focuses on Level A software where structural coverage is being performed on the source code. It does not address structural coverage at the object code level, since that topic is being addressed in another paper.

5. DISCUSSION. To determine the traceability and verification activities and data needed to show adherence to the guidelines in section 6.4.4.2b of DO-178B/ED-12B, an understanding of the information related to structural coverage analysis is required. The provisions of 6.4.4.2b are applicable only to software assured to Level A.

a. Purpose of structural coverage analysis: Section 6.4.4.2 of DO-178B/ED-12B states the objective of structural coverage analysis is to determine which code structures were not exercised by the requirements-based tests. For more information on the purpose of structural coverage consult FAQ #43 of DO-248B/ED-94B. At least one purpose of this provision is to ensure that the behavior of all the programming statements have been examined in an execution environment. For Level C software, DO-178B/ED-12B

considers demonstration of statement coverage, data coupling, and control coupling sufficient. For Level B software, demonstration of decision coverage, statement coverage, data coupling, and control coupling is considered adequate. For level A software, DO-178B/ED-12B recommends modified condition/decision coverage (MC/DC) in addition to statement coverage, decision coverage, data coupling, and control coupling. The glossary of DO-178B/ED-12B provides definitions for each of these coverage criteria.

b. Summary of structural coverage analysis objectives: The summary of the objectives dealing with structural coverage are contained in DO-178B/ED-12B, Annex A, Table 7, objectives 5 through 8. The demonstration of these structural coverage criteria for software levels A, B, and C may be done on the source code. However, additional verification is required for Level A code *"… if the compiler generates object code that is not directly traceable to the source code."* (For another perspective on why this is required see DP #12 in DO-248B/ED-94B.) For Level A software, the two main issues are how to identify object code that the compiler generates that is not "directly traceable" to the source code and how to determine what additional verification of this untraceable object code is needed. For the remainder of this paper, these issues are discussed as the "traceability issue" and the "verification issue," respectively.

c. Identifying compiler-inserted object code: Section 4.4.2b of DO-178B/ED-12B provides some insight into what is meant by the "traceability issue." This section is repeated for reference:

> "To implement certain features, compilers for some languages may produce object code that is not directly traceable to the source code, for example, initialization, built-in error detection or exception handling (subparagraph 6.4.4.2, item b). The software planning process should provide a means to detect this object code and to ensure verification coverage and define the means in the appropriate plan."

(1) As can be established from the above extract from DO-178B/ED-12B, the purpose of the traceability exercise is to identify compiler-added code that is not visible at or traceable to the source code level. Some examples of such code mentioned in DO-178B/ED-12B are:

- initializations (4.4.2b),
- built-in error detection (4.4.2b),
- exception handling (4.4.2b), and
- compiler-generated array-bound check (6.4.4.2b).

(2) Any traceability activity or method needs to demonstrate that it is capable of detecting and identifying added code. Some programming languages may contain features that make this detection of added code difficult (for example, optimizations such as register usage and loop unrolling or use of polymorphism in C++).

(3) In general, some of the object code implementing these types of program language features may not be traceable directly to the source code. In some cases, "hidden" libraries are included at link time. Such libraries should be examined. This traceability exercise is not intended to demonstrate that every object code statement is correct, but only intended to detect and identify the presence of additional, untraceable object code. In all cases, the functionality of this additional code should be clearly identified, so that its behavior can be understood.

> Note: Any assembly libraries that are developed by an applicant will likely not result in traceability issues, if they were developed under the same guidelines as the operational code. However, all libraries should be examined.

d. Verification of code that is not directly traceable: Once this additional, untraceable code has been identified as described in the previous paragraphs, section 6.4.4.2b, specifies that *"…additional verification should be performed on the object code to establish the correctness of such generated code sequences …"*. DO-178B/ED-12B specifically used the term "verification" to permit review and analysis, as well as testing, to demonstrate the correctness of this added code. The additional verification activity needs to demonstrate that the added code identified during the traceability exercise above correctly implements its functionality and does not introduce anomalous behavior in the operational software.

e. Clarifying a misconception: A common misconception exists that if software is Level A and data cannot be produced that demonstrates that the compiler generated object code is directly traceable to the source code statements, then modified condition/decision coverage will have to be performed on the object code in order to satisfy objective 7 in DO-178B/ED-12B Table A-7. In contrast with this misconception, the DO-178B/ED-12B objective is the verification of "correctness" or acceptable behavior within the execution context of the program of any code added by the compiler. "Correctness" is in this case not to be equated to achieving modified condition/decision coverage. Although the applicant may choose to do the structural coverage on the object code provided certain provisions have been satisfied as explained in FAQs #43 and #42 of DO-248B/ED-94B, the identification and verification of the added code is still needed.

f. Compiler optimization: Compiler optimization is another area addressed under section 4.4.2a of DO-178B/ED-12B. This involves the analytical determination that the optimization features do not compromise the ability of the test cases to demonstrate requirements-based testing and structural coverage consistent with the software level. This is a separate issue from the traceability and additional verifications issues addressed by Section 4.4.2b. This is outside the scope of this paper.

g. Summary of source code to object code traceability analysis: To summarize, there are three steps that an applicant needs to address in performing source code to object code traceability and verification:

1. detect and identify any added object code;
2. establish the functionality of that code; and
3. verify the correctness of the added, untraceable code sequences.

h. Related information: Other related information on structural coverage can be found in DO-248B/ED-94B, including: DP#3 (The Differences Between DO-178A/ED-12A and DO-178B/ED-12B Guidance for Meeting the Objective of Structural Coverage), DP #8 (Structural Coverage and Safety Objectives), and FAQ #44 (Why is structural testing not a DO-178B/ED-12B requirement?).

6. <u>PROCEDURES</u>. For any project involving the interpretation of DO-178B/ED-12B section 6.4.4.2b and the related objective 7 in Annex A Table 7, the certification authority (or designee, if applicable) should follow the procedures listed in this section, when reviewing Level A projects.

a. The certification authority (or designee, if applicable) should determine that the source code to object code traceability and verification approach has been described in the software plans (typically the Plan for Software Aspects of Certification or the Software Verification Plan or both), for Level A software where MC/DC is being performed on the source code.

b. The certification authority (or designee, if applicable) should ensure that the applicant (or supplier) is following the approved software plans.

c. For Level A software, the certification authority (or designee, if applicable) should ensure that an analysis has been performed by the applicant (or their supplier) to detect and identify any additional object code in the airborne software application generated by the compiler, the linker, libraries, the run-time system, the operating system, or any other means that is not directly traceable to the source code statements.

d. The certification authority (or designee, if applicable) should establish that the functionality of any additional code was summarized and documented. These analyses may take many forms. Some example forms are described in section 6.h below.

e. For any analysis, the certification authority (or designee, if applicable) should ensure that the applicant (or their supplier) performed the analyses on representative source code and object code that is applicable to the target environment of the intended airborne system. The compiler options for the analysis should be the same as the compiler options used to develop the airborne software.

f. The certification authority (or designee, if applicable) should ensure that the applicant (or their supplier) has considered any used features of the programming language, as well as hardware and architecture specific features of the system (e.g., memory addressing implemented by the linker, run-time system, operating system, or other means). Any representative code used for the identification and behavior analysis of added, untraceable object code should be evaluated to ensure that they were produced in identical development environment, using identical procedures, configurations, and build instructions as that intended for the software application and target environment of the intended airborne system.

g. If any additional, untraceable object code is detected by this analysis, the certification authority (or designee, if applicable) should examine the verification records associated with this added code. The records should demonstrate what the acceptable behavior of the added code within the execution context of the program is, and how the applicant or their developer verified it. The acceptability of the behavior will be based on the summary of the behavior captured in the analysis documentation. The verification may include testing, analysis, review, or some combination thereof.

h. The following paragraphs describe two potential approaches for detecting, identifying, and establishing the acceptability of the behavior of any code added by the compiler, linker, libraries, run-time system, operating system, or other means (please note that there may be other approaches, particularly as compiler complexity increases):

(1) Manual review/analysis of the complete program - The source code and associated assembly code listings of the object code may be manually examined (reviewed) to detect any code added by the compiler beyond what is required for execution of the source code statements. The certification authority should examine the analysis/review results and some representative source code/object code groups to gain confidence that the analysis/review was done correctly and that the additional, untraceable object code identified correctly implements its required functionality. The acceptability of the verification records should be determined as described in sections 6.c – 6.g above.

(2) Analysis of a complete set of used/implemented programming constructs – The applicant (or their supplier) may elect to impose coding standards throughout the program that might limit the number of programming operations and libraries to a smaller subset of the programming language and compiler-provided library functions. A programming operation is a component of the programming language such as an operator (+, -, etc.) or a logical operation (loops, comparisons, etc.) that the programmer combines within the rules of the language to develop computer programs. If evidence can be provided that the operational program and library functions fully comply with the coding standards, then the certification authority may accept analysis that is done on the subset. Once all of the existing code can be shown to be in compliance with the coding standards, one or more tests combining all of the constructs specified in the coding

standards can be produced and compiled. (Note: This assumes that all types of compiler-inserted code and their interactions can be traced, tracked, and verified; if this is not the case, this approach may not be valid.) The results of these tests are then examined and analyzed. The certification authority should examine the analysis and some source code/object code groups to gain confidence that the analysis was done correctly and that any additional, untraceable code correctly implements its identified functionality. The acceptability of the verification records should be determined as described in sections 6.c – 6.g above. To establish the validity of the test programs, the applicant should document the results of a comparison between the results of the analysis and some representative code from the actual operational program. The representative code should be chosen from complex functions where a higher probability of added, untraceable functionality might exist (for example, software handling arrays, potential hidden calls to libraries, exception handlers, traps or loop constructs). The representative operational code should validate the test program's conclusions. The certification authority should evaluate the choice of representative operational code to ensure that the more complex functions were selected. The comparison analysis should also be examined to ensure that the conclusions of the test program analysis are applicable to and valid for the operational program.

> NOTE: Other approaches may be acceptable but should be coordinated with the appropriate certification authority specialists. Any and all approaches should be detailed in the verification plan (or other software plans, as appropriate) and coordinated with the approving certification authority.

i. The certification authority should ensure that any software development and operational environmental factors, such as changes to the compiler, linker, libraries, run-time system, or operating system throughout the program have been considered in the reviews, analyses, and tests.

j. The certification authority may give credit for previous analysis, if the applicant can demonstrate the applicability to the current source code, object code, compiler, library, linker, run-time system, operating system, hardware, system and software architecture, and operational environment.